Linear Programming - Report

Making the A Matrix

In the given problem, every state has 5 different values which are stored in tuples. The tuple is of the form <Position, Materials, Arrows, MM_state, MM_health> which represents each state.

The range of the following tuples is given by :

```
POSITIONS = ["W", "N", "E", "S", "C"]
MATERIALS = [0, 1, 2]
ARROWS = [0, 1, 2, 3]
MM = ["R", "D"]
HEALTH = [0, 25, 50, 75, 100]
```

Therefore we have 600 different states. We get the total number of states by the get_tot_action _function.

```
def get_tot_actions(self):
        tot_actions = 0
        for name,s in self.states.items():
            tot_actions = tot_actions + len(s.actions)
        return tot_actions
```

Therefore we get 1936 different actions. The indices of the A matrix are [total_states, total_actions].

The matrix is populated as follows:

- We iterate over all the possible states (all possible combinations of the tuple defined above).Each state has been mapped to an index using its variable "name" as its key.

```
for name, s in self.states.items():
            self.name_idx_map.update({name : idx})
            idx+=1
```

- For every unique state, we consider all the possible actions and transitions from that state. For every possible action.
- For example, if an action A1 has a probability p of changing state from A to B, we add p to the A[index(A)][index(action A1)] and -p to A[index(B)][index(action A1)]. For actions that lead terminal states, we add 1 to the respective cell.
- This way, the A matrix essentially captures the probabilistic flow between state transitions caused by an action. Code for generating A is as follows:

```
idx = 0
        i = 0
        for name, s in self.states.items():
            for a_name, action in s.actions.items():
                if action.name == "NONE":
                    a[i][idx] += 1
                    idx += 1
                    break

                for trans in action.transitions:
                    dest_idx = self.getidx_of_state(trans.dest)
                    a[i][idx] += trans.prob
                    a[dest_idx][idx] -= trans.prob

                idx += 1
            i += 1

        start_idx = self.getidx_of_state(self.init_state.name)
```

Finding Policy

- A policy essentially defines the flow of choice of actions to find the maximum reward. It is represented by a sequence of STATE-ACTION pairs.
- The X vector that we calculate using the LP solver defines the policy for the MDP. We store the utility of the STATE - ACTION pair in the X vector.
- In order to calculate X, we generate two more vectors R and alpha. The R vector is used to hold the reward of taking a valid action from each state. The code for generating R is as follows:

```
def get_r(self):
        r = np.zeros((1, self.dim)) #change here
```

```
        idx = 0
        for name, s in self.states.items():
            for action_name, action in s.actions.items():
                if action_name == "NONE":
                    r[0][idx] = 0
                    idx += 1
                    break

                for trans in action.transitions:
                    r[0][idx] += trans.prob * trans.reward
                idx+=1
        #print(f"num_actions {idx}")
        return r
```

- The alpha vector stores the initial probabilities of the agent starting in every state. For our problem, we know the start state hence we initialize the alpha vector with 1 at the index of the starting state and rest all zeros.
- A linear program is an optimization technique with contains:
  - Decision Variables (X)
  - Inequality Constraints (Ax = alpha)
  - Objective Function (RX)
- Therefore we get our LPP as:

```
maximize RX
      subject to AX  = alpha,
                 X >= 0
```

- After we obtain the X vector, we iterate over all the states. For each state, we iterate over all the X values associated with the state and we add the ACTION associated to the highest X value to the policy.

Multiple Policies?

Yes, multiple policies can exist as for one state, actions with the same X value are interchangeable.

- Currently, if there are many actions that have the highest X values, we take the action represented by the minimum index. (np.argmax()). We

can choose the action represented by the highest index by iterating as $x[i] >= x[i+1]$ over all the relevant indices.

- We can also shuffle the ACTIONS list to consider actions in a different order. Note that this will not change any values in the R, A, and, alpha matrix but it could change the mapping of actions done on the indices. Even if we do that, we observe that we get the same final vector for X only that the indices representing the STATE-ACTION pair have been shuffled around.
- In conclusion , as long as we are choosing values with the highest X values at every step, it doesn't matter which ACTION-STATE pair we choose to use in our policy; the final reward we get in each case will be same proving multiple policies do exist.